CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Dagstuhl Research Online Publication Server

Oracle-Based Primal-Dual Algorithms for Packing and Covering Semidefinite Programs

Khaled Elbassioni
Khalifa University of Science and Technology, Masdar City Campus,
P.O. Box 54224, Abu Dhabi, UAE
khaled.elbassioni@ku.ac.ae

Kazuhisa Makino
Research Institute for Mathematical Sciences (RIMS), Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract

Packing and covering semidefinite programs (SDPs) appear in natural relaxations of many combinatorial optimization problems as well as a number of other applications. Recently, several techniques were proposed, that utilize the particular structure of this class of problems, to obtain more efficient algorithms than those offered by general SDP solvers. For certain applications, such as those described in this paper, it maybe required to deal with SDP's with *exponentially* or *infinitely* many constraints, which are accessible only via an *oracle*. In this paper, we give an efficient primal-dual algorithm to solve the problem in this case, which is an extension of a *logarithmic-potential* based algorithm of Grigoriadis, Khachiyan, Porkolab and Villavicencio (SIAM Journal of Optimization 41 (2001)) for packing/covering linear programs.

2012 ACM Subject Classification Mathematics of computing → Semidefinite programming; Theory of computation → Numeric approximation algorithms

Keywords and phrases Semidefinite programs, packing and covering, logarithmic potential, primal-dual algorithms, approximate solutions

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.43

Related Version A full version of the paper is available at `https://arxiv.org/abs/1809.09698`.

1 Introduction

1.1 Packing and Covering SDPs

We denote by $\mathbb{S}^n$ the set of all $n \times n$ real symmetric matrices and by $\mathbb{S}^n_+ \subseteq \mathbb{S}^n$ the set of all $n \times n$ positive semidefinite matrices. Consider the following pairs of *packing-covering* semidefinite programs (SDPs):

$$\begin{array}{lll} & z^*_I = \max \quad C \bullet X & \text{(P-I)} \\ \text{s.t.} & A_i \bullet X \le b_i, \forall i \in [m] & \\ & X \in \mathbb{R}^{n\times n},\ X \succeq 0 & \end{array} \qquad \begin{array}{lll} & z^*_I = \min \quad b^T y & \text{(C-I)} \\ \text{s.t.} & \sum_{i=1}^m y_i A_i \succeq C & \\ & y \in \mathbb{R}^m,\ y \ge 0 & \end{array}$$

$$\begin{array}{lll} & z^*_{II} = \min \quad C \bullet X & \text{(C-II)} \\ \text{s.t.} & A_i \bullet X \ge b_i, \forall i \in [m] & \\ & X \in \mathbb{R}^{n\times n},\ X \succeq 0 & \end{array} \qquad \begin{array}{lll} & z^*_{II} = \max \quad b^T y & \text{(P-II)} \\ \text{s.t.} & \sum_{i=1}^m y_i A_i \preceq C & \\ & y \in \mathbb{R}^m,\ y \ge 0 & \end{array}$$


© Khaled Elbassioni and Kazuhisa Makino;
licensed under Creative Commons License CC-BY

where $C, A_1, \ldots, A_m \in \mathbb{S}^n_+$ are (non-zero) positive semidefinite matrices, and $b = (b_1, \ldots, b_n)^T \in \mathbb{R}^m_+$ is a nonnegative vector. In the above, $C \bullet X := \mathrm{Tr}(CX) = \sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij}$, and "$\succeq$" is the *Löwner order* on matrices: $A \succeq B$ if and only if $A - B$ is positive semidefinite. This type of SDPs arise in many applications, see, e.g. [19, 18] and the references therein.

We will make the following assumption throughout the paper:

(A) $b_i > 0$ and hence $b_i = 1$ for all $i \in [m]$.

It is known that, under assumption (A), *strong duality* holds for problems (P-I)-(C-I) (resp., (P-II)-(C-II)).

Let $\epsilon \in (0, 1]$ be a given constant. We say that (X, y) is an ϵ-*optimal* primal-dual solution for (P-I)-(C-I) if (X, y) is a primal-dual feasible pair such that

$$C \bullet X \geq (1 - \epsilon)b^T y \geq (1 - \epsilon)z^*_I. \tag{1}$$

Similarly, we say that (X, y) is an ϵ-optimal primal-dual solution for (P-II)-(C-II) if (X, y) is a primal-dual feasible pair such that

$$C \bullet X \leq (1 + \epsilon)b^T y \leq (1 + \epsilon)z^*_{II}. \tag{2}$$

Since in this paper we allow the number of constraints m in (P-I) (resp., (C-II)) to be *exponentially* (or even infinitely) large, we will assume the availability of the following *oracle*:

Max(Y) (resp., **Min**(Y)): Given $Y \in \mathbb{S}^n_+$, find $i \in \mathrm{argmax}_{i\in[m]} A_i \bullet Y$ (resp., $i \in \mathrm{argmin}_{i\in[m]} A_i \bullet Y$).

Note that an *approximation* oracle computing the maximum (resp., minimum) above within a factor of $(1 - \epsilon)$ (resp., $(1 + \epsilon)$) is also sufficient for our purposes.

Our objective in this paper is to develop oracle-based *primal-dual* algorithms that find ϵ-optimal solutions for (P-I)-(C-I) and (P-II)-(C-II). An interesting property of our algorithms which distinguishes them from most previously known algorithms is that they produce solutions which are *sparse*, in the following sense: A primal-dual solution (X, y) to (C-I) (resp., (P-II)) is said to be η-*sparse*, if the size of $\mathrm{supp}(y) := \{i \in [m] : y_i > 0\}$ is at most η. Two applications for SDP's with infinite/exponential number of constraints are given in Section 3.

1.2 Main Result and Related Work

Problems (P-I)-(C-I) and (P-II)-(C-II) can be solved using general SDP solvers, such as interior-point methods: for instance, the barrier method (see, e.g., [29]) can compute a solution, within an *additive* error of ϵ from the optimal, in time $O(\sqrt{n}m(n^3 + mn^2 + m^2)\log\frac{1}{\epsilon})$ (see also [1, 34]). However, due to the special nature of (P-I)-(C-I) and (P-II)-(C-II), better algorithms can be obtained. Most of the improvements are obtained by using *first order methods* [4, 6, 7, 2, 13, 19, 20, 21, 22, 28, 30, 31], or second order methods [17, 18]. In general, we can classify these algorithms according to whether they:

(I) are *width-independent*: the running time of the algorithm depends *polynomially* on the bit length of the input; for example, in the of case of (P-I)-(C-I), the running time is $\mathrm{poly}(n, m, \mathcal{L}, \log\tau, \frac{1}{\epsilon})$, where $\mathcal{L}$ is the maximum bit length needed to represent any number in the input; on the other hand, the running time of a width-dependent algorithm will depend polynomially on a"width parameter" ρ, which is polynomial in $\mathcal{L}$ and τ;

(II) are *parallel*: the algorithm takes $\mathrm{polylog}(n, m, \mathcal{L}, \log\tau) \cdot \mathrm{poly}(\frac{1}{\epsilon})$ time, on a $\mathrm{poly}(n, m, \mathcal{L}, \log\tau, \frac{1}{\epsilon})$ number of processors;

(III) *output sparse solutions*: the algorithm outputs an η-sparse solution to (C-I) (resp., (P-II)), for $\eta = \text{poly}(n, \log m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$ (resp., $\eta = \text{poly}(n, \log m, \mathcal{L}, \frac{1}{\epsilon})$), where τ is a parameter that bounds the trace of any optimal solution X (see Section 2 for details);

(IV) are *oracle-based*: the only access of the algorithm to the matrices $A_1, \ldots, A_m$ is via the maximization/minimization oracle, and hence the running time is independent of m.

Table 1 gives a summary[1] of the most relevant results together with their classifications, according to the four criteria described above. We note that almost all these algorithms for packing/covering SDP's are generalizations of similar algorithms for packing/covering linear programs (LPs), and most of them are essentially based on an *exponential potential function* in the form of *scalar exponentiation*, e.g., [4, 22], or *matrix exponentiation* [6, 7, 2, 21, 19]. For instance, several of these results use the scalar or matrix versions of the *multiplicative weights updates* (MWU) method (see, e.g., [5]), which are extensions of similar methods for packing/covering LPs [14, 15, 35, 32].

In [16], a different type of algorithm was given for covering LPs (indeed, more generally, for a class of concave covering inequalities) based on a *logarithmic* potential function. In this paper, we show that this approach can be extended to provide oracle-based algorithms for both versions of packing and covering SDPs:

▶ **Theorem 1.** *For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-I)-(C-I), outputs an $O(n\mathcal{L}\log(n\tau) + \frac{n}{\epsilon^2})$-sparse $O(\epsilon)$-optimal primal-dual pair in time*[2] *$\tilde{O}\big(\frac{n^{\omega+1}\mathcal{L}\log\tau}{\epsilon^{2.5}} + \frac{n\mathcal{L}\mathcal{T}\log\tau}{\epsilon^2}\big)$, where $\mathcal{T}$ is the time taken by a single call to the oracle Max$(\cdot)$ and ω is the exponent of matrix multiplication.*

▶ **Theorem 2.** *For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-II)-(C-II), outputs an $O(n\mathcal{L}\log n + \frac{n}{\epsilon^2})$-sparse $O(\epsilon)$-optimal primal-dual pair in time $\tilde{O}\big(\frac{n^{\omega+1}\mathcal{L}\log n}{\epsilon^{2.5}} + \frac{n\mathcal{L}\mathcal{T}}{\epsilon^2}\big)$, where $\mathcal{T}$ is the time taken by a single call to the oracle Min$(\cdot)$.*

As we can see from the table, among all the algorithms listed, the logarithmic-potential algorithm, presented in this paper, is the only one that produces sparse solutions, in the sense described above. Moreover, the only known other oracle-based algorithm (matrix Matrix MWU [7]) is *not* width-independent.It can also be shown (see [12]) that a modified version of the matrix exponential MWU algorithm [6] can yield sparse solutions for (P-II)-(C-II). However, the overall running time of this matrix MWU algorithm is larger by a factor of (roughly) $\Omega(n^{3-\omega})$ than that of the logarithmic-potential algorithm. Moreover, we were not able to extend the matrix MWU algorithm to solve (P-I)-(C-I) (in particular, it seems tricky to bound the number of iterations).

A work that is also related to ours is the sparsification of graph Laplacians [8] and positive semidefinite sums [33]. Given matrices $A_1, \ldots, A_m \in \mathbb{S}^n_+$ and $\epsilon > 0$, it was shown in [33] that one can find, in $O\big(\frac{n}{\epsilon^2}(n^\omega + \mathcal{T})\big)$ time, a vector $y \in \mathbb{R}^m_+$ with support size $O(\frac{n}{\epsilon^2})$, such that $B \preceq \sum_i y_i A_i \preceq (1+\epsilon)B$, where $B := \sum_i A_i$ and $\mathcal{T}$ is the time taken by a single call to the minimization oracle Min(Y) (for a not necessarily positive semidefinite matrix Y). An immediate corollary is that, given an ϵ-optimal solution y for (C-I) (resp., (P-I)), one can find in $O\big(\frac{n}{\epsilon^2}(n^\omega + \mathcal{T})\big)$ time an $O(\epsilon)$-optimal solution y' with support size $O(\frac{n}{\epsilon^2})$. Interestingly, the algorithm in [33] (which is an extension for the rank-one version in [8]) uses the *barrier potential function* $\Phi'(x, F) := \text{Tr}\big((H - xI)^{-1}\big)$ (resp., $\Phi'(x, H) := \text{Tr}\big((xI - H)^{-1}\big)$), while in

[1] We provide rough estimates of the bounds, as some of them are not explicitly stated in the corresponding paper in terms of the parameters we consider here.

[2] $\tilde{O}(\cdot)$ hides polylogarithmic factors in n and $\frac{1}{\epsilon}$.

Table 1 Different Algorithms for Packing/covering SDPs.

Paper	Problem	Technique	Most Expensive Operation	# Iterations	Width-indep.	Parallel	Sparse	Oracle-based
[4, 22]	(P-I) (C-II)	MWU	max / min eigenvalue of a PSD matrix $\tilde{O}(\frac{n^2}{\epsilon})$	$O(\frac{\rho \log m}{\epsilon^2})$	No	No	No*)	No
[7]	(P-I) (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{\rho^2 r^2 \log n}{\epsilon^2 (z_I^*)^2})$	No	No	No*)	Yes
[17, 19]	(P-I)	Nesterov's smoothing technique [27, 28]	Matrix exponentiation $O(n^3)$	$O(\frac{\tau \log m}{\epsilon})$	No	No	No	No
[18]	(C-II)	Nesterov's smoothing technique [27, 28]	min eigenvalue of a non PSD matrix $O(n^3)$	$O(\frac{\rho^2 \log(nm)}{\epsilon})$	No	No	No	No
[20]	(P-I)& (C-II)	MWU technique [27, 28]	eigenvalue decomposition $O(n^3)$	$O(\frac{\log^{13} n \log m}{\epsilon^{13}})$	Yes	Yes	No	No
[30, 31]	(P-II)& (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{\log^3 m}{\epsilon^3})$	Yes	Yes	No	No
[2]	(P-I)& (C-II)	Gradient Descent + Mirror Descent	Matrix exponentiation $O(n^3)$	$O(\frac{\log^2(mn) \log \frac{1}{\epsilon}}{\epsilon^2})$	Yes	Yes	No	No
[12]	(P-II)& (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{n \log n}{\epsilon^2})$	Yes	No	Yes	Yes
This paper	(P-II) & (C-II) (P-II) & (C-II)	Logarithmic potential [16]	Matrix inversion $O(n^\omega)$	$O(n \log(n\mathcal{L}\tau) + \frac{n}{\epsilon^2})$ $O(n \log(n/\epsilon) + \frac{n}{\epsilon^2})$	Yes	No	Yes	Yes

*) In fact, these algorithms find sparse solutions, in the sense that the dependence of the size of the support of the dual solution on m is at most logarithmic; however, the dependence of the size of the support on the bit length $\mathcal{L}$ is not polynomial.

our algorithms (generalizing the potential function in [16]) we use the logarithmic potential function $\Phi(x, H) = \ln x + \frac{\epsilon}{n} \ln \det \left(H - xI\right) = \ln x - \frac{\epsilon}{n} \int_x \Phi'(x, H) dx$ (resp., $\Phi(x, H) = \ln x - \frac{\epsilon}{n} \ln \det \left(xI - H\right) = \ln x - \frac{\epsilon}{n} \int_x \Phi'(x, H) dx$). Sparsification algorithms with better running times were recently obtained in [3, 24]. Since the sparse solutions produced by our algorithms may have support size slightly more (by polylogarithmic factors) than $O(\frac{n}{\epsilon^2})$, we may use, in a post-processing step, the sparsfication algorithms, mentioned above, to convert the solutions obtained by Theorems 1 and 2 to ones with support size $O(\frac{n}{\epsilon^2})$, without increasing the overall asymptotic running time.

In Section 4, we give an outline of the algorithm and sketch the proof of Theorem 1; the proof of Theorem 2 is similar. To motivate our algorithms, in Section 3, we give two applications that require finding sparse solutions for a packing/covering SDP [3].

2 Reduction to Normalized Form

When $C = I = I_n$, the identity matrix in $\mathbb{R}^{n \times n}$ and $b = \mathbf{1}$, the vector of all ones in $\mathbb{R}^m$, we say that the packing-covering SDPs are in *normalized* form:

$$\begin{aligned} & z_I^* = \max \quad I \bullet X && \text{(N-P-I)} \\ \text{s.t.} \quad & A_i \bullet X \leq 1, \forall i \in [m] \\ & X \in \mathbb{R}^{n \times n},\ X \succeq 0 \end{aligned} \qquad \Bigg| \qquad \begin{aligned} & z_I^* = \min \quad \mathbf{1}^T y && \text{(N-C-I)} \\ \text{s.t.} \quad & \sum_{i=1}^m y_i A_i \succeq I \\ & y \in \mathbb{R}^m,\ y \geq 0. \end{aligned}$$

[3] As pointed out by an anonymous reviewer, solution-sparsity and oracle-access to the input can also be thought of as a way of reducing the *space requirement* of the algorithm, see [23].

$$\begin{array}{lll} & z^*_{II} = \min \quad I \bullet X & \text{(N-C-II)} \\ \text{s.t.} & A_i \bullet X \geq 1, \forall i \in [m] & \\ & X \in \mathbb{R}^{n\times n},\ X \succeq 0 & \end{array} \quad \Bigg| \quad \begin{array}{lll} & z^*_{II} = \max \quad \mathbf{1}^T y & \text{(N-P-II)} \\ \text{s.t.} & \sum_{i=1}^m y_i A_i \preceq I & \\ & y \in \mathbb{R}^m,\ y \geq 0. & \end{array}$$

It can be shown[4] that, at the loss of a factor of $(1+\epsilon)$ in the objective, any pair of packing-covering SDPs of the form (P-I)-(C-I) can be brought in $O(n^3)$, increasing the oracle time only by $O(n^\omega)$, where ω is the exponent of matrix multiplication, to the normalized form (N-P-I)-(N-C-I), under the following assumption:

(B-I) There exist r matrices, say $A_1, \ldots, A_r$, such that $\bar{A} := \sum_{i=1}^r A_i \succ 0$. In particular, $\mathrm{Tr}(X) \leq \tau := \frac{r}{\lambda_{\min}(\bar{A})}$ for any optimal solution X for (P-I).

Similarly, one can show (see [12, 20]) that, at the loss of a factor of $(1+\epsilon)$ in the objective, any pair of packing-covering SDPs of the form (P-II)-(C-II) can be brought in $O(n^3)$ time, increasing the oracle time only by $O(n^\omega)$, to the normalized form (N-P-II)-(N-C-II). Moreover, we may assume in this normalized form that

(B-II) $\lambda_{\min}(A_i) = \Omega\big(\frac{\epsilon}{n} \cdot \min_{i'} \lambda_{\max}(A_{i'})\big)$ for all $i \in [m]$,

where, for a positive semidefinite matrix $B \in \mathbb{S}^{n\times n}_+$, we denote by $\{\lambda_j(B) :\ j = 1, \ldots, n\}$ the eigenvalues of B, and by $\lambda_{\min}(B)$ and $\lambda_{\max}(B)$ the minimum and maximum eigenvalues of B, respectively. With an additional $O(mn^2)$ time, we may also assume that:

(B-II′) $\frac{\lambda_{\max}(A_i)}{\lambda_{\min}(A_i)} = O\big(\frac{n^2}{\epsilon^2}\big)$ for all $i \in [m]$.

Thus, from now on we focus on the normalized problems.

3 Applications

3.1 Robust Packing and Covering SDPs

Consider a packing-covering pair of the form (P-I)-(C-I) or (P-II)-(C-II). In the framework of *robust optimization* (see, e.g. [9, 10]), we assume that each constraint matrix A_i is not known exactly; instead, it is given by a convex uncertainty set $\mathcal{A}_i \subseteq \mathbb{S}^n_+$. It is required to find a (near)-optimal solution for the packing-covering pair under the *worst-case* choice $A_i \in \mathcal{A}_i$ of the constraints in each uncertainty set. A typical example of a *convex* uncertainty set is given by an *affine perturbation* around a nominal matrix $A^0_i \in \mathbb{S}^n_+$:

$$\mathcal{A}_i = \left\{ A_i := A^0_i + \sum_{r=1}^k \delta_r A^r_i :\ \delta = (\delta_1, \ldots, \delta_k) \in \mathcal{D} \right\}, \tag{3}$$

where $A^1_i, \ldots, A^k_i \in \mathbb{S}^n_+$, and $\mathcal{D} \subseteq \mathbb{R}^k_+$ can take, for example, one of the following forms:

- *Ellipsoidal uncertainty*: $\mathcal{D} = E(\delta_0, D) := \{\delta \in \mathbb{R}^k_+ : (\delta - \delta_0)^T D^{-1} (\delta - \delta_0) \leq 1\}$, for given positive definite matrix $D \in \mathbb{S}^k_+$ and vector $\delta_0 \in \mathbb{R}^k_+$ such that $E(\delta_0, D) \subseteq \mathbb{R}^k_+$;
- *Polyhedral uncertainty*: $\mathcal{D} := \{\delta \in \mathbb{R}^k_+ : D\delta \leq w\}$, for given matrix $D \in \mathbb{R}^{h\times k}$ and vector $w \in \mathbb{R}^h$.

[4] In fact, unlike previously known reductions, the reduction we give in [12] is simpler to compute, as it is based on the *LDL-decompositions* rather than the *eigenvalue* decompositions of the input matrices.

Without loss of generality, we consider the robust version of (N-P-I)-(N-C-I), where A_i, for $i \in [m]$, belongs to a convex uncertainty set $\mathcal{A}_i$. Then the robust optimization problem and its dual can be written as follows:

$$\begin{array}{lll} & z_P^* = \max \quad I \bullet X & \text{(R-P-I)} \\ \text{s.t.} & A_i \bullet X \le 1, \quad \forall A_i \in \mathcal{A}_i \quad \forall i \in [m] & \\ & X \in \mathbb{R}^{n\times n},\ X \succeq 0 & \end{array} \qquad \begin{array}{ll} z_D^* = \inf & \sum_{i=1}^m \int_{\mathcal{A}_i} y^i_{A_i} dA_i \\ & \text{(R-C-I)} \\ \text{s.t.} & \sum_{i=1}^m \int_{\mathcal{A}_i} y^i_{A_i} A_i dA_i \succeq I \\ & y^i \text{ is a } \textit{discrete measure} \text{ on } \mathcal{A}_i, \quad \forall i \in [m]. \end{array}$$

As before, we assume (B-I), where $A_1, \ldots, A_r \in \bigcup_{i\in[m]} \mathcal{A}_i$. We call a pair of solutions (X, y) to be ϵ-optimal for (R-P-I)-(R-C-I), if

$$z_P^* \ge I \bullet X \ge (1-\epsilon) \sum_{i=1}^m \int_{\mathcal{A}_i} y^i_{A_i} dA_i \ge (1-\epsilon) z_D^*.$$

Note that the number of constraints in (R-P-I) is *infinite* and hence any algorithm that solves the problem would have to be oracle-based. The Ellipsoid method is one such algorithm; a more efficient procedure is given by the following corollary of Theorem 1.

▶ **Theorem 3.** *For any $\epsilon > 0$, there is a a randomized algorithm that outputs an $O(\epsilon)$-optimal primal-dual pair for (*R-P-*I)-(*R-C-*I) in time $\tilde{O}\big(\frac{n^{\omega+1}\log\psi}{\epsilon^{2.5}} + \frac{n\mathcal{T}\log\psi}{\epsilon^2}\big)$, where $\psi := \frac{r\cdot\max_{i\in[m], A_i\in\mathcal{A}_i}\lambda_{\max}(A_i)}{\lambda_{\min}(\bar{A})}$ and $\mathcal{T}$ is the time to compute, for a given $Y \in \mathbb{S}^n_+$, a pair (i, A_i) such that*

$$(i, A_i) \in \operatorname{argmax}_{i\in[m],\ A_i\in\mathcal{A}_i} A_i \bullet Y. \tag{4}$$

Note that (4) amounts to solving a linear optimization problem over a convex set. Moreover, for simple uncertainty sets, such as boxes or ellipsoids, such computation can be done very efficiently.

3.2 Carr-Vempala-Type Decomposition

Consider a maximization (resp., minimization) problem over a discrete set $\mathcal{S} \subseteq \mathbb{Z}^n$ and a corresponding SDP-relaxation over $\mathcal{Q} \subseteq \mathbb{S}^n_+$:

$$\begin{array}{ll} z^*_{CO} = \left\{\begin{array}{l}\max\\ \min\end{array}\right\} & C \bullet qq^T \\ q \in \mathcal{S} & \end{array} \qquad \text{(DOP)} \qquad \begin{array}{ll} z^*_{SDP} = \left\{\begin{array}{l}\max\\ \min\end{array}\right\} & C \bullet Q \\ & \text{(SDP-RLX)} \\ Q \in \mathcal{Q}, & \end{array}$$

where $C \in \mathbb{S}^n_+$.

▶ **Definition 4.** *For $\alpha \in (0, 1]$ (resp., $\alpha \ge 1$), an α-integrality gap verifier $\mathcal{A}$ for (SDP-RLX) is a polytime algorithm that, given any $C \in \mathbb{S}^n_+$ and any $Q \in \mathcal{Q}$ returns a $q \in \mathcal{S}$ such that $C \bullet qq^T \ge \alpha C \bullet Q$ (resp., $C \bullet qq^T \le \alpha C \bullet Q$).*

For instance, if $\mathcal{S} = \{-1, 1\}^n$ and $Q = \{X \in \mathbb{S}^n_+ : X_{ii} = 1 \ \ \forall i \in [n]\}$, then a $\frac{2}{\pi}$-integrality gap verifier for the maximization version of (SDP-RLX) is known [26].

Carr and Vempala [11] gave a decomposition theorem that allows one to use an α-integrality gap verifier for a given LP-relaxation of a combinatorial maximization (resp., minimization) problem, to decompose a given fractional solution to the LP into a convex combination of integer solutions that is dominated by (resp., dominates) α times the fractional solution. Here we derive a similar result for SDP relaxations:

▶ **Theorem 5.** *Consider a combinatorial maximization (resp., minimization) problem (DOP) and its SDP relaxation (SDP-RLX), admitting an α-integrality gap verifier $\mathcal{A}$. Assume the set $\mathcal{S}$ is full-dimensional and let $\epsilon > 0$ be a given constant. Then there is a polytime algorithm that, for any given $Q \in \mathcal{Q}$, finds a set $\mathcal{X} \subseteq \mathcal{S}$ of size $|\mathcal{X}| = O(\frac{n^3}{\epsilon^2}\log(nW))$ (resp., of size $|\mathcal{X}| = O(n\log\frac{n}{\epsilon} + \frac{n}{\epsilon^2})$), where $W := \max_{q\in\mathcal{S},\ i\in[n]} |q_i|$, and a set of convex multipliers $\{\lambda_q \in \mathbb{R}_+ :\ q \in \mathcal{X}\}$, $\sum_{q\in\mathcal{X}} \lambda_q = 1$, such that*

$$(1 - O(\epsilon))\alpha Q \preceq \sum_{q\in\mathcal{X}} \lambda_q qq^T \qquad \textit{(resp., } (1 + O(\epsilon))\alpha Q \succeq \sum_{q\in\mathcal{X}} \lambda_q qq^T\textit{).} \tag{5}$$

The proof of Theorem 5 is obtained by considering the following pairs of packing and covering SDPs (of types I and II, respectively):

$$z_I^* = \min \sum_{q\in\mathcal{S}} \lambda_q \qquad \text{(CVX-I)}$$
$$\text{s.t.} \sum_{q\in\mathcal{S}} \lambda_q qq^T \succeq \alpha Q \tag{6}$$
$$\sum_{q\in\mathcal{S}} \lambda_q \geq 1 \tag{7}$$
$$\lambda \in \mathbb{R}^{\mathcal{S}},\ \lambda \geq 0$$

$$z_I^* = \max \quad \alpha Q \bullet Y + u \qquad \text{(CVX-dual-I)}$$
$$\text{s.t.} \quad qq^T \bullet Y + u \leq 1, \forall q \in \mathcal{S} \tag{8}$$
$$Y \in \mathbb{S}^n_+,\ u \geq 0.$$

$$z_{II}^* = \max \sum_{q\in\mathcal{S}} \lambda_q \qquad \text{(CVX-II)}$$
$$\text{s.t.} \sum_{q\in\mathcal{S}} \lambda_q qq^T \preceq \alpha Q \tag{9}$$
$$\sum_{x\in\mathcal{S}} \lambda_q \leq 1 \tag{10}$$
$$\lambda \in \mathbb{R}^{\mathcal{S}},\ \lambda \geq 0$$

$$z_{II}^* = \min \quad \alpha Q \bullet Y + u \qquad \text{(CVX-dual-II)}$$
$$\text{s.t.} \quad qq^T \bullet Y + u \geq 1, \forall q \in \mathcal{S} \tag{11}$$
$$Y \in \mathbb{S}^n_+,\ u \geq 0.$$

It can be shown, using the fact that the SDP relaxation admits an α-integrality gap verifier, that $z_I^* = z_{II}^* = 1$, and that the two primal-dual pairs can be solved in polynomial time using the Ellipsoid method. A more efficient (but approximate version) can be obtained using the algorithms of Theorems 1 and 2. Note that, once we have a set $\mathcal{X}$ as in Theorem 5, its support can be reduced to $O(\frac{n^2}{\epsilon})$ using the sparsification techniques of [8, 33].

4 A Logarithmic Potential Algorithm for (P-I)-(C-I)

In this section we give an algorithm for finding a sparse $O(\epsilon)$-optimal primal-dual solution for (N-P-I)-(N-C-I). Since the algorithm updates only one component of the dual solution in each iteration, it follows that the number of positive components of the dual solution when the algorithm terminates is exactly equal to the number of iterations; from this sparsity follows.

4.1 High-level Idea of the Algorithm

The idea of the algorithm is quite intuitive. It can be easily seen that problem (N-C-I) is equivalent to finding a convex combination of the A_i's that maximizes the minimum eigenvalue, that is, $\max_{y\in\mathbb{R}^m_+:\mathbf{1}^T y=1} \lambda_{\min}(F(y))$, where $F(y) := \sum_{i=1}^m y_i A_i$, and $\mathbf{1}$ is the m-dimensional vector of all ones. Since $\lambda_{\min}(F(y))$ is not a *smooth* function in y, it is more convenient to work with a smooth approximation of it, which is obtained by maximizing (over x) a *logarithmic potential function* $\Phi(x, F(y))$ that captures the constraints that each eigenvalue of $F(y)$ is at least x. The unique maximizer $x = \theta^*$ of $\Phi(x, F(y))$ defines a set of "weights" (these are the eigenvalues of the primal matrix X computed in line 6 of the algorithm) such that the weighted average of the $\lambda_j(F(y))$'s is a very close approximation of $\lambda_{\min}(F(y))$. Thus, to maximize this average (which is exactly $X \bullet F(y)$), we obtain a direction (line 7) along which y is modified with an appropriate step size (line 10). For numbers $x \in \mathbb{R}_+$ and $\delta \in (0, 1)$, a δ-(lower) approximation x_δ of x is a number such that $(1-\delta)x \le x_\delta < x$. For $i \in [m]$, $\mathbf{1}_i$ denotes the ith unit vector of dimension m.

The algorithm is shown as Algorithm 1. The main while-loop (step 4) is embedded within a sequence of scaling phases, in which each phase starts from the vector $y(t)$ computed in the previous phase and uses double the accuracy. The algorithm stops when the scaled accuracy ε_s drops below the desired accuracy $\epsilon \in (0, 1/2)$. When referring to an arbitrary iteration of the algorithm, we assume it is iteration t in phase s.

4.2 Analysis

4.2.1 High-level Idea of the Analysis

The analysis is based on a matrix generalization of the scalar arguments given in [16] (as is the case for all algorithms for SDP's, which are driven from their LP counterparts; see, e.g., [1]). Besides the technical details, the algorithm also requires estimating the minimum eigenvalue of the dual matrix $F(y(t))$, which is done using Lanczos' algorithm (see Section 4.2.4 for details).

The proof of ϵ-optimality follows easily from the stopping condition in line 4 of the algorithm, the definition of the "approximation error" ν in line 8, and the fact that $X \bullet F(y)$ is a very close approximation of $\lambda_{\min}(F(y(t)))$. The main part of the proof is to bound the number of iterations in the inner while-loop (line 4). This is done by using a *potential function argument*: we define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$ and show in Claim 23 that, in each iteration, the choice of the step size in line 9 guarantees that $\Phi(t)$ is increased substantially; on the other hand, by Claim 24, the potential difference cannot be very large, and the two claims together imply that we cannot have many iterations.

4.2.2 Some Preliminaries

Up to Claim 26, we fix a particular iteration s of the outer while-loop in the algorithm. For simplicity in the following, we will sometimes write $F := F(y(t))$, $\theta := \theta(t)$, $\theta^* := \theta^*(t)$, $X := X(t)$, $\hat{F} := A_{i(t)}$, $\tau := \tau(t+1)$, $\nu := \nu(t+1)$, $F' := F(y(t+1))$, and $\theta' := \theta(t+1)$, when the meaning is clear from the context. For $H \succ 0$ and $x \in (0, \lambda_{\min}(H))$, define the logarithmic potential function [16, 29]:

$$\Phi(x, H) = \ln x + \frac{\varepsilon_s}{n} \ln\det\big(H - xI\big). \tag{12}$$

Note that the term $\ln\det\big(H - xI\big)$ forces the value of x to stay away from the "boundary" $\lambda_{\min}(H)$, while the term $\ln x$ pushes x towards that boundary; hence, one would expect the maximizer of $\Phi(x, H)$ to be a good approximation of $\lambda_{\min}(H)$ (see Claim 8).

Algorithm 1 Logarithmic-potential Algorithm for (P-I)-(C-I).

$s \leftarrow 0$; $\varepsilon_0 \leftarrow \frac{1}{2}$; $t \leftarrow 0$; $\nu(0) \leftarrow 1$; $y(0) \leftarrow \frac{1}{r}\sum_{i=1}^{r} \mathbf{1}_i$
while $\varepsilon_s > \epsilon$ **do**
  $\delta_s \leftarrow \frac{\varepsilon_s^3}{32n}$
  while $\nu(t) > \varepsilon_s$ **do**
    $\theta(t) \leftarrow \theta^*(t)_{\delta_s}$, where $\theta^*(t)$ is the smallest positive number root of the equation $\frac{\varepsilon_s \theta}{n}\mathrm{Tr}(F(y(t)) - \theta I)^{-1} = 1$
    $X(t) \leftarrow \frac{\varepsilon_s \theta(t)}{n}(F(y(t)) - \theta(t) I)^{-1}$ /* Set the primal solution */
    $i(t) \leftarrow \mathrm{argmax}_i \, A_i \bullet X(t)$ /* Call the maximization oracle */
    $\nu(t+1) \leftarrow \frac{X(t) \bullet A_{i(t)} - X(t) \bullet F(y(t))}{X(t) \bullet A_{i(t)} + X(t) \bullet F(y(t))}$ /* Compute the error */
    $\tau(t+1) \leftarrow \frac{\varepsilon_s \theta(t) \nu(t+1)}{4n(X(t) \bullet A_{i(t)} + X(t) \bullet F(y(t)))}$ /* Compute the step size */
    $y(t+1) \leftarrow (1 - \tau(t+1))y(t) + \tau(t+1)\mathbf{1}_{i(t)}$ /* Update the dual solution */
    $t \leftarrow t + 1$
  end
  $\varepsilon_{s+1} \leftarrow \frac{\varepsilon_s}{2}$
  $s \leftarrow s + 1$
end
$\hat{X} \leftarrow \frac{(1-\varepsilon_{s-1})X(t-1)}{(1+\varepsilon_{s-1})^2 \theta(t-1))}$; $\hat{y} \leftarrow \frac{y(t-1)}{\theta(t-1)}$ /* Scale primal and dual to retain feasibility */
return $(\hat{X}, \hat{y}, t)$

▷ Claim 6. If $F(y(t)) \succ 0$, then $\theta^*(t) = \mathrm{argmax}_{0<x<\lambda_{\min}(F)} \Phi(x, F(y(t)))$ and $X(t) \succ 0$.

For $x \in (0, \lambda_{\min}(F))$, let $g(x) := \frac{\varepsilon_s x}{n}\mathrm{Tr}(F - xI)^{-1}$. The following claim shows that our choice of δ_s guarantees that $g(\theta)$ is a good approximation of $g(\theta^*) = 1$.

▷ Claim 7. $g(\theta(t)) \in (1 - \varepsilon_s, 1)$.

The following two claims show that $\theta(t) \approx X(t) \bullet F(y(t))$ provides a good approximation for $\lambda_{\min}(F(y(t)))$.

▷ Claim 8. $(1 - \varepsilon_s)\lambda_{\min}(F(y(t))) < \theta(t) < \frac{\lambda_{\min}(F(y(t)))}{1+\varepsilon_s/n}$ and $\frac{\lambda_{\min}(F(y(t)))}{1+\varepsilon_s} \leq \theta^*(t) \leq \frac{\lambda_{\min}(F(y(t)))}{1+\varepsilon_s/n}$.

▷ Claim 9. $\theta(t) < X(t) \bullet F(y(t)) < (1 + \varepsilon_s)\theta(t)$.

Throughout the algorithm, we maintain the invariants that the (non-scaled) dual objective $\mathbf{1}^T y(t)$ is exactly 1, that the step size $\tau(t)$ and the approximation error $\nu(t)$ are between 0 and 1, and that the dual matrix $F(y(t)) = \sum_i y_i(t) A_i$ is positive definite. This is summarized in the following claims.

▷ Claim 10. $\mathbf{1}^T y(t) = 1$.

▷ Claim 11. For all iterations t, except possibly the last, $\nu(t+1), \tau(t+1) \in (0, 1)$.

▷ Claim 12. $F(y(t)) \succ 0$.

4.2.3 Primal Dual Feasibility and Approximate Optimality

Let $t_f + 1$ be the value of t when the algorithm terminates and $s_f + 1$ be the value of s at termination. For simplicity, we write $s = s_f$.

▷ Claim 13. (Primal feasibility). $\hat{X} \succ 0$ and $\max_i A_i \bullet \hat{X} \leq 1$.

Proof. The first claim is immediate from Claim 6. To see the second claim, we use the definition of $\nu(t_f)$ and the termination condition in line 4 (which is also satisfied even if $X(t_f) \bullet A_{i(t_f)} - X(t_f) \bullet F(y(t_f)) = 0$):

$$\frac{X(t_f) \bullet A_{i(t_f)} - X(t_f) \bullet F(y(t_f))}{X(t_1) \bullet A_{i(t_f)} + X(t_f) \bullet F(y(t_f))} \leq \varepsilon_s.$$

$$\therefore (1+\varepsilon_s) X(t_f) \bullet F(y(t_f)) \geq (1-\varepsilon_s) X(t_f) \bullet A_{i(t_f)}$$

$$= (1-\varepsilon_s) \max_i X(t_f) \bullet A_i$$

(by the defintition of $i(t_f)$)

$$\therefore (1+\varepsilon_s)^2 \theta(t_f) \geq (1-\varepsilon_s) \max_i X(t_f) \bullet A_i.$$

($\because X(t_f) \bullet F(y(t_f)) \leq (1+\varepsilon_s)\theta(t_f)$ by Claim 9)

The claim follows by the definition of $\hat{X}$ in step 16 of the algorithm. ◁

▷ Claim 14. (Dual feasibility). $\hat{y} \geq 0$ and $F(\hat{y}) \succ I$.

Proof. The fact that $\hat{y} \geq 0$ follows from the initialization of $y(0)$ in step 1, Claim 11, and the update of $y(t+1)$ in step 10. For the other claim, we have

$$\lambda_{\min}\big(F(\hat{y})\big) = \frac{1}{\theta(t_f)} \lambda_{\min}\big(F(y(t_f))\big) \geq 1 + \frac{\varepsilon_s}{n}. \qquad \text{(by Claim 8)}$$

◁

▷ Claim 15. (Approximate optimality). $I \bullet \hat{X} \geq \left(\frac{1-\varepsilon_s}{1+\varepsilon_s}\right)^2 \mathbf{1}^T \hat{y}$.

Proof. By Claim 7, we have $\mathrm{Tr}(X(t_f)) \geq 1 - \varepsilon_s$, and by Claim 10, we have $\mathbf{1}^T y(t_f) = 1$. The claim follows by the definition of $\hat{X}$ and $\hat{y}$ in step 16. ◁

4.2.4 Running Time per Iteration

Given $F := F(y(t)) \succ 0$, we first compute an approximation $\widetilde{\lambda}$ of $\lambda_{\min}(F)$ using Lanczos' algorithm with a random start [25].

▶ **Lemma 16** ([25]). *Let $M \in \mathbb{S}^n_+$ be a positive semidefinite matrix with N non-zeros and $\gamma \in (0,1)$ be a given constant. Then there is a randomized algorithm that computes, with high (i.e., $1 - o(1)$) probability a unit vector $v \in \mathbb{R}^n$ such that $v^T M v \geq (1-\gamma)\lambda_{\max}(M)$. The algorithm takes $O\big(\frac{\log n}{\sqrt{\gamma}}\big)$ iterations, each requiring $O(N)$ arithmetic operations.*

By Claim 8, we need $\widetilde{\lambda}$ to lie in the range $[\frac{\lambda_{\min}(F)}{1+\varepsilon_s/n}, \lambda_{\min}(F)]$. To obtain $\widetilde{\lambda}$, we may apply the above lemma with $M := F^{-1}$ and $\gamma := \frac{\varepsilon_s}{2n}$. Then in $O\big(\sqrt{\frac{n}{\varepsilon_s}} \log n\big)$ iterations we get $\widetilde{\lambda} := \frac{1-\gamma}{v^T F^{-1} v}$ satisfying our requirement. However, we can save (roughly) a factor of $\sqrt{n}$ in

the running time by using, instead, $M := F^{-n}$ and $\gamma := \frac{\varepsilon_s}{2}$. Let v be the vector obtained from Lemma 16, and set $\widetilde{\lambda} := \left(\frac{1-\gamma}{v^T F^{-n} v}\right)^{1/n}$. Then, as $\lambda_{\max}(M) \geq v^T M v \geq (1-\gamma)\lambda_{\max}(M)$, and $\lambda_{\min}(F) = \lambda_{\max}(F^{-n})^{-1/n}$, we get

$$\frac{\lambda_{\min}(F)}{1+\varepsilon_s/n} \leq (1-\gamma)^{1/n}\lambda_{\min}(F) \leq \widetilde{\lambda} \leq \lambda_{\min}(F). \tag{13}$$

Note that we can compute F^{-n} in $O(n^\omega \log n)$, where w is the *exponent of matrix multiplication*. Thus, the overall running time for computing $\widetilde{\lambda}$ is $O(n^\omega \log n + \frac{n^2 \log n}{\sqrt{\varepsilon_s}})$.

Given $\widetilde{\lambda}$, we know by Claim 8 and (13) that $\theta^*(t) \in [\frac{\widetilde{\lambda}}{1+\varepsilon_s}, \widetilde{\lambda}]$. Then we can apply binary search to find $\theta(t) := \theta^*(t)_{\delta_s}$ as follows. Let $\theta_k = \frac{\widetilde{\lambda}}{1+\varepsilon_s}(1+\delta_s)^k$, for $k = 0, 1, \ldots, K := \lceil \frac{2\ln(1+\varepsilon_s)}{\delta_s} \rceil$, and note that $\theta_L \geq \widetilde{\lambda}$. Then we do binary search on the exponent $k \in \{0, 1 \ldots, K\}$; each step of the search evaluates $g(\theta_k) := \frac{\varepsilon_s \theta_\ell}{n} \mathrm{Tr}(F - \theta_k I)^{-1}$, and depending on whether this value is less than or at least 1, the value of k is increased or decreased, respectively. The search stops when the search interval $[\ell, u]$ has $u \leq \ell + 1$, in which case we set $\theta(t) = \theta_\ell$; the number of steps until this happens is $O(\log K) = O(\log \frac{1}{\delta_s}) = O(\log \frac{n}{\varepsilon_s})$. By the monotonicity of $g(x)$ (in the interval $[0, \lambda_{\min}(F)]$), and the property of binary search, we know that $\theta^* \in [\theta_\ell, \theta_u]$. Thus, by the stopping criterion,

$$\theta(t) = \theta_\ell \leq \theta^*(t) \leq \theta_u \leq \theta_{\ell+1} = (1+\delta_s)\theta_\ell,$$

implying that $(1-\delta_s)\theta^*(t) \leq \theta(t) \leq \theta^*(t)$. Since evaluating $g(\theta_\ell)$ takes $O(n^\omega)$, the overall running time for the binary search procedure is $O(n^\omega \log \frac{n}{\varepsilon_s})$, and hence the total time needed for for computing $\theta(t)$ is $O(n^\omega \log \frac{n}{\epsilon} + \frac{n^2 \log n}{\sqrt{\epsilon}})$.

All other steps of the algorithm inside the inner while-loop can be done in $O(\mathcal{T} + n^2)$ time, where $\mathcal{T}$ is the time taken by a single call to the oracle $\mathrm{Max}(X(t))$ in step 7 of the algorithm. Thus, in view of Claim 26 on the number of iterations below, we obtain Theorem 1.

4.2.5 Number of Iterations

Define $B = B(t) := \frac{n}{\varepsilon_s \theta}\left(\tau X^{1/2}(\hat{F} - F)X^{1/2} - (\theta^* - \theta)X\right)$. The following (technical) claims are needed for the proofs of Claims 23 and 24 below (which are, in turn, the main claims needed for the analysis of the potential function). They can be skipped at a first reading and recalled when needed.

▷ Claim 17. $(F - \theta^* I)^{-1} = \left(\frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta)X\right)^{-1} X$.

▷ Claim 18. $F' - \theta^* I = (F - \theta I)^{1/2}(I + B)(F - \theta I)^{1/2}$.

▷ Claim 19. $\max_j |\lambda_j(B)| \leq \frac{1}{2}$.

▷ Claim 20. $\theta^*(t) < \lambda_{\min}(F(y(t+1)))$.

▷ Claim 21. if $\nu > \varepsilon_s$, then $\mathrm{Tr}(B) \geq \frac{\nu^2}{8}$.

▷ Claim 22. If $\nu > \varepsilon_s$, then $\mathrm{Tr}(B^2) < \frac{\nu^2}{10}$.

Define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$.

The following claim states that the potential difference between two consecutive iterations of the algorithm is sufficiently large.

▷ Claim 23. For t, $t+1$ in phase s, $\Phi(t+1) - \Phi(t) \geq \frac{\varepsilon_s \nu(t+1)^2}{40n}$.

Proof. Note that Claim 20 implies that θ^* is feasible to the problem $\max\{\Phi(\xi, F') : 0 \leq \xi \leq \lambda_{\min}(F')\}$. Thus,

$$\begin{aligned}
\Phi(t+1) = \Phi(\theta^*(t+1), F') &\geq \ln\theta^* + \frac{\varepsilon_s}{n}\ln\det(F' - \theta^* I). \\
\therefore \Phi(t+1) - \Phi(t) &\geq \frac{\varepsilon_s}{n}\left(\ln\det(F' - \theta^* I) - \ln\det(F - \theta^* I)\right) \\
&\geq \frac{\varepsilon_s}{n}\left(\ln\det\left(F' - \theta^* I\right) - \ln\det(F - \theta I)\right) && (\because \theta \leq \theta^*) \\
&= \frac{\varepsilon_s}{n}\ln\det\left(I + B\right) && \text{(by Claim 18)} \\
&= \frac{\varepsilon_s}{n}\sum_{j=1}^{n}\ln\left(1 + \lambda_j(B)\right) \\
&\geq \frac{\varepsilon_s}{n}\sum_{j=1}^{n}\left(\lambda_j(B) - \lambda_j(B)^2\right) \\
& && \text{(by Claim 19 and } \ln(1+z) \geq z - z^2, \forall z \geq -0.5) \\
&= \frac{\varepsilon_s}{n}\left(\mathrm{Tr}(B) - \mathrm{Tr}(B^2)\right) \\
&> \frac{\varepsilon_s}{8n}\nu^2 - \frac{\varepsilon_s}{10n}\nu^2 && \text{(by Claims 21 and 22)} \\
&= \frac{\varepsilon_s}{40n}\nu^2.
\end{aligned}$$

◁

On the other hand, the following claim states that the overall potential difference between any iterations cannot be too large.

▷ Claim 24. For any t, t' in phase s,

$$\Phi(t') - \Phi(t) \leq (1+\varepsilon_s)\ln\frac{X(t)\bullet A_{i(t)}}{(1-\varepsilon_s)X(t)\bullet F(y(t))}.$$

Proof. Write $F = F(y(t))$, $\theta^* := \theta^*(t)$, $\theta := \theta(t)$, $X := X(t)$, $F' = F(y(t'))$, $\theta'^* := \theta^*(t')$. Then

$$\begin{aligned}
\Phi(t') - \Phi(t) &= \ln\frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n}\ln\det\left[(F - \theta^* I)^{-1}(F' - \theta'^* I)\right] \\
&= \ln\frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n}\ln\det\left[\left(\frac{\varepsilon_s\theta}{n}I - (\theta^* - \theta)X\right)^{-1}X(F' - \theta'^* I)\right] && \text{(by Claim 17)} \\
&= \ln\frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n}\left[\ln\det\left(\frac{\varepsilon_s\theta}{n}I - (\theta^* - \theta)X\right)^{-1} + \ln\det\left[X(F' - \theta'^* I)\right]\right] \\
&\leq \ln\frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n}\left[\ln\left(\frac{\varepsilon_s\theta}{n} - \frac{\delta_s\theta}{1-\delta_s}\right)^{-n} + \ln\det\left[X(F' - \theta'^* I)\right]\right] \\
& && (\because \mathrm{Tr}(X) \leq 1 \text{ by Claim 7 and } (1-\delta_s)\theta^* \leq \theta)
\end{aligned}$$

$$\leq \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n}\left[\ln\left(\frac{n}{(1-\varepsilon_s)\varepsilon_s\theta}\right)^n + \ln\det X(F' - \theta'^* I)\right]$$

(by defintion of δ_s)

$$= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \frac{\varepsilon_s}{n}\ln\left[\det X(F' - \theta'^* I)\right]$$

$$= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \frac{\varepsilon_s}{n}\sum_{j=1}^{n}\ln\lambda_j\big(X(F' - \theta'^* I)\big)$$

$$\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\left(\frac{1}{n}\sum_{j=1}^{n}\lambda_j\big(X(F' - \theta'^* I)\big)\right)$$

(by the concavity of $\ln(\cdot)$)

$$= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\left(\frac{1}{n}\mathrm{Tr}(XF' - \theta'^* X)\right)$$

$$\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\left(\frac{X \bullet F' - \theta'^*(1-\varepsilon_s)}{n}\right)$$

($\because \mathrm{Tr}(X) \geq 1 - \varepsilon_s$ by Claim 7)

$$= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\big(X \bullet F' - \theta'^*(1-\varepsilon_s)\big)$$

$$\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\left(\max_{y\in\mathbb{R}^m_+:\ \mathbf{1}^T y=1} X \bullet F(y) - \theta'^*(1-\varepsilon_s)\right)$$

($\because \mathbf{1}^T y(t') = 1$ by Claim 10)

$$= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\big(X \bullet A_{i(t)} - \theta'^*(1-\varepsilon_s)\big)$$

(by defintion of $i(t)$)

$$\leq \max_{0\leq\xi<X\bullet A_{i(t)}}\left\{\ln\frac{\xi}{(1-\varepsilon_s)\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln\big(X \bullet A_{i(t)} - \xi\big)\right\}$$

$$= (1+\varepsilon_s)\ln\frac{X \bullet A_{i(t)}}{(1-{\varepsilon_s}^2)\theta} + \ln\frac{\theta}{\theta^*} \qquad (\max(\cdot) \text{ is achieved at } \xi = \tfrac{X\bullet A_{i(t)}}{1+\varepsilon_s})$$

$$\leq (1+\varepsilon_s)\ln\frac{X \bullet A_{i(t)}}{(1-{\varepsilon_s}^2)\theta} \qquad (\because \theta \leq \theta^*)$$

$$\leq (1+\varepsilon_s)\ln\frac{X \bullet A_{i(t)}}{(1-\varepsilon_s)X \bullet F}. \qquad \text{(by Claim 9)}$$

◁

Recall by assumption (B-I) that $\bar{A} := \sum_{i=1}^{r} A_i \succ 0$.

▷ Claim 25. $\frac{X(0)\bullet A_{i(0)}}{X(0)\bullet F(y(0))} \leq \psi := \frac{r\cdot\lambda_{\max}(A_{i(0)})}{\lambda_{\min}(\bar{A})} \leq \frac{r\cdot\max_i\lambda_{\max}(A_i)}{\lambda_{\min}(\bar{A})} \leq n\tau 2^{\mathcal{L}}$.

Proof. Let $X(0) = \sum_{j=1}^{n}\lambda_j u_j u_j^T$ be the spectral decomposition of $X(0)$. Then,

$$X(0) \bullet A_{i(0)} = \sum_{j=1}^{n}\lambda_j A_{i(0)} \bullet u_j u_j^T \leq \sum_{j=1}^{n}\lambda_j\lambda_{\max}(A_{i(0)}) = \lambda_{\max}(A_{i(0)})\cdot\mathrm{Tr}(X(0))$$

$$X(0) \bullet F(y(0)) = \sum_{j=1}^{n}\lambda_j F(y(0)) \bullet u_j u_j^T \geq \frac{1}{r}\sum_{j=1}^{n}\lambda_j\lambda_{\min}(\bar{A}) = \frac{1}{r}\lambda_{\min}(\bar{A})\cdot\mathrm{Tr}(X(0)).$$

The claim follows. ◁

Now we combine claims 23, 24 and 25 to obtain a bound on the number of iterations.

▷ Claim 26. The algorithm terminates in at most $O\left(n \log \psi + \frac{n}{\epsilon^2}\right)$ iterations.

▶ Remark 27. If we do not insist on a sparse dual solution, then we can use the initialization $y(0) \leftarrow \frac{1}{m}\mathbf{1}$ in step 1 in Algorithm 1, and replace ψ in Claim 25, and hence in the running time in Claim 26, by m.

References

1 F. Alizadeh. Interior Point Methods in Semidefinite Programming with Applications to Combinatorial Optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

2 Z. Allen-Zhu, Y. Tat Lee, and L. Orecchia. Using Optimization to Obtain a Width-independent, Parallel, Simpler, and Faster Positive SDP Solver. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1824–1831, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

3 Z. Allen-Zhu, Z. Liao, and L. Orecchia. Spectral Sparsification and Regret Minimization Beyond Matrix Multiplicative Updates. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 237–245, New York, NY, USA, 2015. ACM.

4 S. Arora, E. Hazan, and S. Kale. Fast Algorithms for Approximate Semidefinite Programming using the Multiplicative Weights Update Method. In *Proc. 46th Symp. Foundations of Computer Science (FOCS)*, pages 339–348, 2005.

5 S. Arora, E. Hazan, and S. Kale. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012.

6 S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proc. 39th Symp. Theory of Computing (STOC)*, pages 227–236, 2007.

7 S. Arora and S. Kale. A Combinatorial, Primal-Dual Approach to Semidefinite Programs. *J. ACM*, 63(2):12:1–12:35, May 2016.

8 J. Batson, D. Spielman, and N. Srivastava. Twice-Ramanujan Sparsifiers. *SIAM Review*, 56(2):315–334, 2014.

9 A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

10 A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Math. Program.*, 92(3):453–480, 2002.

11 R. D. Carr and S. Vempala. Randomized metarounding. *Random Struct. Algorithms*, 20(3):343–352, 2002.

12 K. Elbassioni and K. Makino. Finding Sparse Solutions for Packing and Covering Semidefinite Programs. *CoRR*, abs/1809.09698, 2018. `arXiv:1809.09698`.

13 Dan Garber and Elad Hazan. Sublinear Time Algorithms for Approximate Semidefinite Programming. *Math. Program.*, 158(1-2):329–361, July 2016.

14 N. Garg and J. Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

15 M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

16 M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. *SIAM Journal of Optimization*, 41:1081–1091, 2001.

17 G. Iyengar, D. J. Phillips, and C. Stein. Approximation Algorithms for Semidefinite Packing Problems with Applications to Maxcut and Graph Coloring. In Michael Jünger and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization (IPCO)*, pages 152–166, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

18 G. Iyengar, D. J. Phillips, and C. Stein. Feasible and Accurate Algorithms for Covering Semidefinite Programs. In Haim Kaplan, editor, *Algorithm Theory - SWAT 2010*, pages 150–162, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

19 G. Iyengar, D. J. Phillips, and C. Stein. Approximating Semidefinite Packing Programs. *SIAM J. on Optimization*, 21(1):231–268, January 2011.

20 R. Jain and P. Yao. A Parallel Approximation Algorithm for Positive Semidefinite Programming. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 463–471, 2011.

21 R. Jain and P. Yao. A parallel approximation algorithm for mixed packing and covering semidefinite programs. *CoRR*, abs/1201.6090, 2012. `arXiv:1201.6090`.

22 P. Klein and H.-I Lu. Efficient Approximation Algorithms for Semidefinite Programs Arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 338–347, New York, NY, USA, 1996. ACM.

23 P. N. Klein and H.-I. Lu. Space-Efficient Approximation Algorithms for MAXCUT and COLORING Semidefinite Programs. In K.-Y. Chwa and O. H. Ibarra, editors, *Algorithms and Computation*, pages 388–398, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

24 Y. T. Lee and H. Sun. An SDP-based Algorithm for Linear-sized Spectral Sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 678–687, New York, NY, USA, 2017. ACM.

25 Z. Leyk and H. Woźniakowski. Estimating a largest eigenvector by Lanczos and polynomial algorithms with a random start. *Numerical Linear Algebra with Applications*, 5(3):147–164, 1999.

26 Y. Nesterov. Quality of semidefinite relaxation for nonconvex quadratic optimization. CORE Discussion Papers 1997019, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), March 1997.

27 Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, May 2005.

28 Y. Nesterov. Smoothing Technique and its Applications in Semidefinite Optimization. *Mathematical Programming*, 110(2):245–259, July 2007.

29 Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

30 R. Peng and K. Tangwongsan. Faster and Simpler Width-independent Parallel Algorithms for Positive Semidefinite Programming. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 101–108, New York, NY, USA, 2012. ACM.

31 R. Peng, K. Tangwongsan, and P. Zhang. Faster and Simpler Width-Independent Parallel Algorithms for Positive Semidefinite Programming. *CoRR*, abs/1201.5135, 2016. `arXiv:1201.5135`.

32 S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. In *Proc. 32nd Symp. Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

33 M. K. De Carli Silva, N. J. A. Harvey, and C. M. Sato. Sparse Sums of Positive Semidefinite Matrices. *ACM Trans. Algorithms*, 12(1):9:1–9:17, December 2015.

34 L. Vandenberghe and S. Boyd. Semidefinite Programming. *SIAM Review*, 38(1):49–95, 1996. URL: `http://www.jstor.org/stable/2132974`.

35 N. E. Young. Sequential and Parallel Algorithms for Mixed Packing and Covering. In *Proc. 42nd Symp. Foundations of Computer Science (FOCS)*, pages 538–546, 2001.